ARL 71-0287

AD739890

A POSTERIORI FORWARD ERROR ANALYSIS

NAI-KUAN TSAO

APPLIED MATHEMATICS RESEARCH LABORATORY

DECEMBER 1971

PROJECT 7071

Approved for public release; distribution unlimited.

AEROSPACE RESEARCH LABORATORIES
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
WRIGHT-PATTERSON AIR FORCE BASE, OHIO

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Aerospace Research Laboratories Applied Mathematics Research Laboratory Wright-Patterson AFB, Ohio 45433	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP

3. REPORT TITLE
A Posteriori Forward Error Analysis

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Scientific Interim

5. AUTHOR(S) (First name, middle initial, last name)
Nai-Kuan Tsao

6. REPORT DATE December 1971	7a. TOTAL NO. OF PAGES 23	7b. NO. OF REFS 3

8a. CONTRACT OR GRANT NO. In-House Research	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO. 7071-00-14	
c. DoD Element 61102F	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d. DoD Subelement 681304	ARL 71-0287

10. DISTRIBUTION STATEMENT
Approved for public release; distribution unlimited

11. SUPPLEMENTARY NOTES TECH OTHER	12. SPONSORING MILITARY ACTIVITY Aerospace Research Laboratories (LR) Air Force Systems Command Wright-Patterson AFB, Ohio 45433

13. ABSTRACT

The general principle of a posteriori forward error analysis is discussed. The fundamental idea is simply based on the fact that the difference between the computed result of any of the basic floating-point operations and the exact result can be estimated using the computed result. For algorithms with finite number of arithmetic operations, this idea can be extended easily so that forward error analysis is possible. Some results of certain useful algorithms are derived using this approach.

DD FORM 1 NOV 65 1473

UNCLASSIFIED
Security Classification

UNCLASSIFIED
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Error Analysis						
Floating-point arithmetic						
Analytic and Synthetic operation						
Polynomial						
Linear algebraic equations						

UNCLASSIFIED
Security Classification

ERRATA

ARL 71-0287 December 1971

1. Page 3 - line 2 to line 4:

   CHANGE:

   · , we simply move the factor $(1 + \delta)$ in equation (1.3) to the left-hand side of the equation, thus we have

   TO:

   ···, Lemma 1 can be expressed [4] as:

2. Page [illegible] - Eq. (3.1) should read:

$$\eta_n = fl(x_1 x_2 \ldots x_n)$$

3. Page 18 - Add:

[4] Forsythe, G. E. and C. B. Moler, Computer Solution of Linear Algebraic Systems, Prentice Hall, Englewood Cliffs, New Jersey, 1967.

AEROSPACE RESEARCH LABORATORIES
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
WRIGHT-PATTERSON AIR FORCE BASE, OHIO

FOREWORD

This research was accomplished while the author held a National Research Council Postdoctoral Resident Research Associateship supported by the Applied Mathematics Research Laboratory of the Aerospace Research Laboratories. The author wishes to thank Dr. Paul J. Nikolai for many helpful suggestions concerning the style of the present presentation. Thanks are also due to Mrs. Georgene Graves for typing the manuscript.

ABSTRACT

The general principle of a posteriori forward error analysis is discussed. The fundamental idea is simply based on the fact that the difference between the computed result of any of the basic floating-point operations and the exact result can be estimated using the computed result. For algorithms with finite number of arithmetic operations, this idea can be extended easily so that forward error analysis is possible. Some results of certain useful algorithms are derived using this approach.

TABLE OF CONTENTS

1. Introduction.

In recent years considerable attention has been given to the effect of rounding errors upon the numerical solution of various problems involving algebraic processes. An outstanding contribution on this topic has been made by J.H. Wilkinson [1,2,3]. His backward error analysis shows that the computed results are the exact solutions to a perturbed problem and the bounds for the perturbations can be obtained numerically. In other words, if we are computing a mathematical expression given by

$$y = g(x_1,x_2,\ldots,x_n) \tag{1.1}$$

the backward analysis shows that the computed $\bar{y}$ satisfies exactly a perturbed equation of the form

$$\bar{y} = g(x_1+\varepsilon_1,x_2+\varepsilon_2,\ldots,x_n+\varepsilon_n) \tag{1.2}$$

where ε_i are perturbations whose bounds, in general, could be obtained.

From now on we will only consider normalized floating-point computations with t bits allocated to the mantissa of a floating-point number. In this setting the backward error analysis is based on the repeated use of the following lemma [3]:

Lemma 1. Let * denote any of the operators +, -, x, /. Then

$$fl(x * y) = (x * y)(1 + \delta) \qquad |\delta| \le 2^{-t} = u \tag{1.3}$$

where x and y are any real numbers and fl(x * y) is the correctly rounded result of the floating operation *.

We observe that Lemma 1 is indeed itself the result of backward error analysis. It expresses the result of any floating-point operation as the result of an exact arithmetic operation on slightly perturbed data and the bound for the perturbation is known. Therefore, if the x_i are not known exactly, the backward analysis will enable us to decide whether the solution obtained numerically is as good as the original data warrants by comparing the bounds for the ε_i with the known errors in x_i; however, it does not tell us how much is the difference between the computed $\bar{y}$ and the exact y if the original set of data is regarded as exact. This problem can be solved if a <u>forward</u> error analysis can be carried out to trace the forward propagation of individual rounding errors and then to compare the computed results with those which exact computation would have produced. Furthermore, this analysis is useful only if the difference between the computed results and the exact results is a simple function of the computed results. In other words, a useful analysis should show that

$$\bar{y} - y = \varepsilon(\bar{y}) \tag{1.4}$$

where $\varepsilon(\bar{y})$ is some simple function of $\bar{y}$. The letter requirement is necessary to obtain bounds for the errors $\varepsilon(\bar{y})$.

We will show in this paper that this <u>a posteriori forward error analysis</u> is possible by properly modifying Lemma 1 such that the modified lemma satisfies the requirement expressed in equation (1.4). Some common floating-point operations are then analyzed and the results applied to some specific algorithms.

2. The basic lemma.

To satisfy the requirement stated in (1.4), we simply move the factor $(1 + \delta)$ in equation (1.3) to the left-hand side of the equation, thus we have

Lemma 2. If x and y are two given floating-point numbers, and * is used to denote any of the operators +, -, $\times$, /. Then

$$(1 + \Delta)\, fl(x * y) = x * y, \; 1 + \Delta = \frac{1}{1+\delta}, \; |\Delta| \leq 2^{-t} = u \quad (2.1)$$

Note from Lemma 2 the difference between computed result $fl(x * y)$ and the exact result $x * y$ is $(\Delta)\, fl(x * y)$ which is a function of the computed result. Since most of the computations are carried out by using these operators sequentially, the error at each operation could thus be monitored by the use of this lemma.

In the formulation of Lemma 2, we have treated the division operator / on the same basis as that of addition +, subtraction -, or multiplication $\times$; namely, division is regarded as an independent operation which is distinct from the additive or the multiplicative operators +, -, or $\times$. However, the division could also be carried out by considering alternatively the following problem: namely, for given x and y, an unknown z is sought such that, without actually performing the division $\frac{x}{y}$, we have

$$yz = x \quad (2.2)$$

In this respect we are decomposing x into a product yz. Thus the computational equation corresponding to (2.2) is

$$yz(1 + \Delta') = x \qquad |\Delta'| \leq u \tag{2.3}$$

which can only tell us the difference between the computed decomposition yz and the exact decomposition x. Hence the algorithm used for (2.2) is "analytic" in nature. On the other hand, if division is done to find $z = \frac{x}{y}$, then this operation is "synthetic" as two unknowns x and y are "synthesized" to form z. The addition, subtraction, and multiplication operations can all be considered as "synthetic" in this respect. We will see later that this idea can also be used to classify algorithms and to interpret the results of error analysis.

3. Errors of extended products and sums.

We first consider the extended product p_n defined as

$$p_n = fl(x_1, x_2, \ldots, x_n) \tag{3.1}$$

We assume henceforth that x_i are floating-point numbers and that operations take place in the order in which they are written. We use the following recursive algorithm to evaluate (3.1):

$$p_1 = x_1$$

$$p_{k+1} = fl(p_k x_{k+1}) \qquad k = 1,2,\ldots,n-1. \tag{3.2}$$

Now applying Lemma 2 to (3.2), we have

$$p_1 = x_1$$

$$p_{k+1}(1 + \varepsilon_{k+1}) = p_k x_{k+1} \qquad k = 1,2,\ldots,n-1. \tag{3.3}$$
$$|\varepsilon_{k+1}| \leq u$$

Hence in general we have

$$p_n \prod_{i=2}^{n} (1 + \varepsilon_i) = \prod_{i=1}^{n} x_i \tag{3.4}$$

It can be shown [3] that if n-1 is appreciably smaller that 2^t, then

$$(1 - u)^{n-1} \leq \prod_{i=2}^{n} (1 + \varepsilon_i) = 1 + E \leq (1 + u)^{n-1} \tag{3.5}$$

Thus we have proved the following lemma:

<u>Lemma 3.</u> For an extended product of n floating-point numbers defined in (3.1), we have

$$p_n(1 + E) = \prod_{i=1}^{n} x_i \tag{3.6}$$

Where 1 + E satisfies (3.5).

For an extended sum defined as

$$s_n = fl(\sum_{i=1}^{n} x_i), \tag{3.7}$$

we can similarly define the recursion

$$s_1 = x_1$$

$$s_{k+1} = fl(s_k + x_{k+1}) \qquad k = 1,2,\dots, n-1 \tag{3.8}$$

to carry out the computation. Applying Lemma 2 repeatedly to (3.8), we have the following lemma:

<u>Lemma 4</u>. For an extneded sum of n floating-point numbers defined in (3.7), we have

$$s_n + \varepsilon = \sum_{i=1}^{n} x_i \tag{3.9}$$

where

$$|\varepsilon| \le |\sum_{i=2}^{n} \delta_i s_i| \le u \sum_{i=2}^{n} |s_i| \tag{3.10}$$

We see that the errors generated in the computation of extended product and that of extended sum can each be estimated after the computation by using Lemmas 3 and 4 respectively. Observe that in extended product the upper bound for the relative error term E is independent of the computation order as is shown in (3.5). On the other hand, the absolute error in the evaluation of extended sum does depend on the order they are added. If all x_i are of the same sign, then from (3.10) the upper bound for the absolute error $|\varepsilon|$ is smallest if the terms are added in order of increasing magnitude. Furthermore, if the x_i are of different signs, then it is also advisable to prearrange x_i such that they are in the order of increasing magnitude with alternate signs.

4. Applications

Applying the previous lemmas to the analysis of algorithms for inner product evaluation, polynominal evaluation and matrix decomposition, we have the following results:

Theorem 1 (Inner Product Evaluation). Let the inner product defined as

$$t = fl(\sum_{i=1}^{n} a_i b_i) \tag{4.1}$$

be computed by the following algorithm:

Step A. Compute

$$c_i = fl(a_i b_i) \qquad i = 1,2,\ldots,n. \tag{4.2}$$

Step B. Compute

$$t_1 = c_1$$

$$t_{k+1} = fl(t_k + c_{k+1}) \qquad k = 1,2,\ldots,n. \tag{4.3}$$

$$t = t_n$$

Then we have

$$t + \epsilon_t = \sum_{i=1}^{n} a_i b_i \tag{4.4}$$

where

$$|\epsilon_t| \leq u \; [\sum_{i=1}^{n} |c_i| + \sum_{i=z}^{n} |t_i|] \tag{4.5}$$

Proof. Applying Lemma 2 to (4.2) and (4.3), we have

$$c_i(1 + \Delta_i) = a_i b_i \qquad |\Delta| \le u,\ i = 1,2,\dots,n \tag{4.6}$$

$$t_1 = c_1$$

$$t_{k+1}(1 + \delta_{k+1}) = t_k + c_{k+1},\ |\delta_{k+1}| \le u, \quad k = 1,2,\dots,n-1. \tag{4.7}$$

$$t = t_n$$

Adding (4.6) and (4.7) for all i and k respectively, we have

$$\sum_{i=1}^{n} c_i + \sum_{i=1}^{n} c_i\Delta_i = \sum_{i=1}^{n} a_i b_i \tag{4.8}$$

$$t + \sum_{k=1}^{n-1} t_{k+1}\delta_{k+1} = \sum_{i=1}^{n} c_i \tag{4.9}$$

Combining (4.8) and (4.9), we have finally

$$t + \varepsilon_t = \sum_{i=1}^{n} a_i b_i \tag{4.10}$$

Where $$\varepsilon_t = \sum_{i=2}^{n} t_i\delta_i + \sum_{i=1}^{n} c_i\delta_i \tag{4.11}$$

Clearly (4.11) satisfies (4.5) and thus the proof is now complete.

We see the format of the error in Theorem 1 is similar to that in Lemma 4. Therefore the comments regarding computation order for extended sum are also applicable in the present case.

Theorem 2. (Nested Polynomial Evaluation). To evaluate a polynomial defined as

$$p(x) = fl[a_0 x^n + a_1 x^{n-1} + \dots + a_n], \tag{4.12}$$

let the following nested algorithm be applied:

$$\begin{aligned} b_0 &= a_0 \\ c_{k+1} &= fl(b_k x) \qquad k = 0,1,2,\dots,n-1 \\ b_{k+1} &= fl(c_{k+1} + a_{k+1}) \\ p(x) &= b_n \end{aligned} \tag{4.13}$$

Then we have

$$p(x) + \sum_{i=1}^{n} \varepsilon_i x^{n-i} = \sum_{i=0}^{n} a_i x^{n-i} \tag{4.14}$$

where

$$|\varepsilon_i| \leq u\, [\, |b_i| + |c_i|\,], \quad i = 1,2,\dots,n \tag{4.15}$$

<u>Proof</u>. Applying Lemma 2 to (4.13), we have

$$\begin{aligned} b_0 &= a_0 \\ c_{k+1} &= b_k x - c_{k+1}\varepsilon'_{k+1}, \qquad |\varepsilon'_{k+1}| \leq u \\ b_{k+1} &= c_{k+1} + a_{k+1} - b_{k+1}\delta'_{k+1}, \quad |\delta'_{k+1}| \leq u \quad k = 0,1,2,\dots,n-1. \\ p(x) &= b_n \end{aligned} \tag{4.16}$$

Simplifying, we have

$$b_0 = a_0$$

$$b_{k+1} = b_k x + a_{k+1} - c_{k+1}\delta_{k+1} - b_{k+1}\delta'_{k+1} \qquad k=0,1,2,\ldots,n-1 \tag{4.17}$$

$$p(x) = b_n$$

By repeatedly substituting (4.17) for b_{k+1} starting from $k = n-1$, we have

$$p(x) + \sum_{i=1}^{n} \varepsilon_i x^{n-i} = \sum_{i=0}^{n} a_i x^{n-i} \tag{4.18}$$

where

$$\varepsilon_i = c_i\delta_i + b_i\delta'_i \tag{4.19}$$

The theorem follows from (4.19)

We observe from (4.18) that the error term depends not only on the computed values b_i and c_i, but also depends on the powers of x, which are not explicitly computed and hence are unknowns. Thus, extra computations are needed if error bound is to be estimated.

Theorem 3. (Unnested Polynomial Evaluation). If the polynomial defined in (4.12) is evaluated by the following algorithm:

Step A. Compute

$$y_n = 1$$

$$y_k = fl(y_{k+1}x) \qquad k = n-1, n-2, \ldots, 0. \tag{4.20}$$

Step B. Compute

$$z_n = a_n$$
$$z_j = fl(a_j y_j) \qquad j = n-1, n-2, \ldots, 0 \tag{4.21}$$

Step C. Compute

$$s_n = z_n$$
$$s_i = fl(s_{i+1} + z_i) \qquad i = n-1, n-2, \ldots, 0 \tag{4.22}$$
$$p(x) = s_0 \quad ,$$

then we have

$$p(x) + \sum_{i=0}^{n-1} (1 + \delta_i)\, z_i \Delta_i + \epsilon_p = \sum_{i=0}^{n} a_i x^{n-i} \tag{4.23}$$

where $|\delta_i| \leq u, \quad (1 - u)^{n-i} \leq 1 + \Delta_i \leq (1 + u)^{n-i}$ and

$$|\epsilon_p| \leq u \sum_{i=0}^{n} [\ |z_i| + |s_i|\ ] \tag{4.24}$$

The proof is similar to that of Theorem 2.

We should note that although an extra n multiplications are needed for this unnested algorithm, we do have results which are useful for error estimation. This is shown in (4.23). Furthermore, if extra computations are carried out to estimate the error in (4.14) resulting from the nested algorithm, then the unnested and nested algorithm are equivalent in terms of number of operations.

Theorem 4 (Matrix Decomposition). For the matrix equation

$$Rx = b \tag{4.25}$$

where $R = (r_{ij})$ is an n by n non-singular lower triangular matrix and b is an n-vector, then the component of x can be computed in the order of $x_1, x_2, \ldots, x_n$ by using the following substitution algorithm:

$$x_1 = fl\left(\frac{b_1}{r_{11}}\right)$$

$$x_i = fl\left[\frac{-r_{i1}x_1 - r_{i2}x_2 - \ldots - r_{i,i-1}x_{i-1} + b_i}{r_{ii}}\right], \quad i = 2,3,\ldots,n \tag{4.26}$$

If (4.26) for each x_i is computed in the following sequence:

Step A. Compute

$$y_{ik} = fl(-r_{ik}x_k) \qquad k = 1,2,\ldots,i-1 \tag{4.27}$$

Step B. Compute

$$s_{ij} = fl(y_{i1} + y_{i2} + \ldots + y_{ij}) \qquad j = 1,2,\ldots,i-1 \tag{4.28}$$

Step C. Compute

$$z_i = fl(s_{i,i-1} + b_i)$$
$$x_i = fl\left(\frac{z_i}{r_{ii}}\right) \tag{4.29}$$

then the computed x satisfies

$$Rx + \eta = b \tag{4.30}$$

where $n = (n_i)$ is an error vector such that

$$|\eta_1| \leq u \cdot |r_{11}x_1|$$

$$|n_i| \leq e \cdot |r_{ii}x_i| + |\varepsilon_{i,i-1}| \qquad i = 2,3,\ldots,n \tag{4.31}$$

and $(1 - u)^2 \leq 1 + e \leq (1 + u)^2$, $|\varepsilon_{i,i-1}| \leq u \left[\sum_{k=1}^{i-1} |y_{ik}| + \sum_{k=z}^{i-1} |s_{ik}| \right]$

The proof can be obtained easily by applying Lemmas 2 and 4 to equations (4.27), (4.28) and (4.29).

We observe from (4.30) that if we are to find the error between the computed solution x and the exact solution $R^{-1}b$, then it is easily seen that

$$R^{-1}b - x = R^{-1}\eta \tag{4.32}$$

Thus the error is a function of R^{-1} which, just like the powers of x in nested polynomial evaluation, has never been explicitly computed. Extra computations are therefore necessary for error estimation.

We might argue that the results of Theorem 2 and Theorem 4 do not satisfy the requirement of (1.4) for useful a posteriori error analysis even if Lemma 2 is used consistently in the analysis. However, we should realize that algebraically we have assumed that the powers of x and R^{-1}

are implicitly generated by the algorithms in Theorem 2 and Theorem 4 respectively. Unfortunately, these "efficient" algorithms do not yield these necessary data computationally. Therefore, the "inefficient" unnested algorithm is a "better" algorithm for polynomial evaluation from the point of view of error estimation. Similarly we could conjecture that Cramer's rule might be a "better" algorithm than substitution algorithm for solving triangular system of equations from this respect.

5. Conclusions.

We have seen for certain algorithms the repeated use of Lemma 2 can lead to useful a posteriori results for error estimations. For some "efficient" algorithms the results are not very "useful". This is because that in order to be "efficient" certain steps of computations have to be skipped which result in insufficient data for error estimation. Another explanation is as follows: The nested algorithm in Theorem 2 essentially decomposes the original polynomial $\sum_{i=0}^{n} a_i x^{n-1}$ into a nested product $\{\ldots[(a_0x + a_1)\ x + a_2]x + \ldots\}\ x + a_n$ with the assumption that the powers of x are generated implicitly; similarly the substitution algorithm in Theorem 4 effectively decomposes b into a product Rx. Hence they are "analytic" in nature. If we are asking only how good is the decomposition, then Theorems 2 and 4 do give us "useful" results concerning the difference between computed decomposition and exact decomposition. Hence they are indeed useful a posteriori results. On the other hand, the algorithms used in Theorem 1 and 3 are "synthetic" in nature as results are "synthesized" step by step without taking "efficient" short cuts. These observations are consistent with the basic results of (2.3) where the division $z = \frac{x}{y}$ is considered as an analytic process if z is computed such that $yz = x$ without actually carrying out the division. The distinction between "analytic" and "synthetic" processes is therefore essential in interpreting the results of error analysis. Furthermore, for matrix equations of the type $Ax = b$ where A is a general n by n non-singular matrix, it is well known that the closeness of Ax to b does not necessarily guarantee the closeness of x to $A^{-1}b$; thus it is questionable that we should use "efficient" analytic algorithms instead of using "inefficient" synthetic algorithms if ultimate error estimation is required for the solution of these systems.

REFERENCES

[1] Wilkinson, J.H., "Error Analysis of Floating-Point Computations", Num. Math. 2 (1960), 319-340.

[2] Wilkinson, J.H., "Error Analysis of Direct Methods of Matrix Inversion", J. ACM. 8 (1961), 281-330.

[3] Wilkinson, J.H., Rounding Errors in Algebraic Processes, Prentice Hall, Englewood, New Jersey, 1963.